CHAPTER 1

Collaborative Engineering

Collaborative engineering is an important subject that is currently missing in schools, although some courses include a few of its aspects. I doubt I can cover it all at once, but I will try my best to *integrate* development technology and the development process under one roof, where integration-ready systems can be created.

Technology tends to fragment: to focus on pieces and omit the glue. The development philosophy that I associate with the terms of collaborative engineering helps keep a better balance between a narrow focus on particular components and their multiple connections to the rest of the world. The practice of collaborative engineering also assists in establishing a repeatable—yet flexible and constantly improving—development process. This is the foundation for integration-ready systems development.

What is collaborative engineering all about? It is about the development process in its organization, management, and methodology, integrated with innovative development technologies. I have had the privilege of teaching corporate developers and architects in the United States and overseas. Many times, I have had the amazing feeling that some of my brightest students, experienced developers, knew all the pieces of the puzzle and still could not start putting them together.

For example, almost every programmer knows one of the main design rules: Separate business from presentation logic. However, there are still more cases that break this rule than cases that follow it. One of the hardest questions is how to apply the rule properly.

Some developers believe that although theory is good, it is too costly for real life. This myth prevents many of them from developing the skills (in everyday practice) to quickly apply an existing design pattern or recognize a new one that will solve the problem "once and forever" and allow them to move on to other problems.

Guided by this myth, we win time and money on separate projects but miss the bigger picture, in which the industry pays a high price for overall inefficiency as well as data and service replication.

I think that by teaching separate pieces of the art and science of development, we omit the glue, the philosophy of programming that helps us find the right balance between "do it simply" and "do it once," which sometimes leads to beautiful solutions in which the two rules are not in conflict but are happily married.

Software is built from big and small blocks selected by architects and developers. In the top-to-bottom development pattern, the big blocks are selected first; the developers then try to fill the holes with the smaller blocks. During this process, they often find that the original direction of the design was wrong and there is a need for remodeling.

How can you start walking in the right direction before you can see the end of the road? Even some good developers cannot begin a project until they can see all the details involved. Unfortunately, the scope of enterprise applications and time limits rarely offer this luxury to developers.

The right answer is a development process that has a framework yet is flexible – a process that is understood, followed, and enhanced by developers.

At this point, some readers may be thinking that all this talk about a process is management's piece of the pie and not something developers need to worry about. Let's read further and reserve our judgment. We will see later how much the management style and the development process contribute to overall results.

Management Style as an Important Part of the Development Process: True Leaders versus "Pure" Managers

> I do not even have time to read my own email reminders; of course I missed yours.
>
> —From a management conversation (the terrible truth)

While working for multiple corporations as a developer or one of the key managers, I observed different management styles. In very generic terms, I would distinguish two major tendencies: "pure" management and true leadership. What is the difference between pure managers and real leaders? Some companies hire managers with little or no background in the field they are supposed to supervise. Their role is to provide and track plans and serve as a layer to smooth the edges among groups related to the plan.

However, project management is not just a Microsoft tool. It takes a leader to build a team, create a teamwork process, and promote best practices and collaborative technologies. It takes a leader to make decisions with creativity, not just through multiple-choice answers. Removing unnecessary layers and dealing with leaders instead of managers can greatly benefit projects, teams, and companies. I have learned this the hard way, working on multimillion dollar projects, consulting start-up companies, and teaching (and learning from) enterprise architects.

Shifting the weight of management toward developers enables those developers to grow and become leaders themselves. This approach focuses more on people than on current projects or tasks. It assumes (and this is the right assumption!) that the *main assets of a company are people*—and teams. The best teams produce the best products and services.

Workers (e.g., testers, developers, and system administrators) need to communicate and to translate to managers the essence of the work they are doing.

This takes minutes if a manager is a leader who knows the work. It takes hours for a pure manager who tries to learn a subject on the fly. The next time a worker wants to play a similar song with the manager, he or she has to start from the first music class—this note is called *A*, this is a *C sharp*—again and again. Even then, only a fraction of the information is retained.

Why? The orchestra conductor (the manager) is not a musician at all. (I know that many are, in a more literal sense.) There is no background where information can be stored. A pure manager often feels forced to choose between several opposing voices or lines in a document, and each voice or line sounds and looks the same as the others.

The management process often turns into a game for such a person, a game with the following rules:

- Use buzzwords, preferably generic (e.g., "stay focused!") or at least technological (e.g., "scalable solution!").
- Do not translate words into actions. Actions can be punished, because any action can be less than 100 percent safe, or incomplete, or not quick enough. Words are always right ("stay focused!").

A leader owns the solution that he or she created. *This ownership feeling is at the heart of the development of any creative process*. (I just read that again, and it looks like a line from Ayn Rand.) In this context, *ownership* means responsibility to make and foresee changes, fix problems, and answer all questions related to the solution.

Pure management removes ownership feelings, destroys creativity, and makes people "come and go." Every developer in the team is a leader who owns and has full responsibility for a specific area. Experienced developers with teamwork skills become the best managers or leaders.

On the opposite extreme of this spectrum is a leader who owns all the solutions, thus has a hard time delegating responsibilities. Such leadership can improve product quality in the short term (if this leader is the most experienced developer).

At the same time, this "babysitting" removes the ownership feelings of the others and prevents developers from growing on the job while making all the development tasks a sequential process funneling through a single leader-bottleneck. A leader should trust and delegate.

Product design and project management are interrelated. I recommend a practice in which team leaders are not only responsible for the part of the project delivered by their team, but also personally involved in that part's integration into the overall project. Leaders keep teams and projects together as orchestra conductors do with musicians and music.

Where is the border between a leader and a pure manager? We all have elements of both, in different proportions. We act as pure managers as soon as we start talking about things beyond our expertise without investing time and effort in learning the field.

Being in a position to issue orders makes the pure manager rather dangerous. Investing time in active learning is hard—but rewarding—and develops a leader.

"I have no time to even read my email" is a phrase from a Dilbert book. It is still a very popular line in the management drama. Another extreme is "I have done much more complicated projects before." Phrases like these are used to excuse a person from learning the specifics of a current task or project. Pure managers avoid specifics. This limits

their participation in a development process in which generic frames are filled with specific details.

Development Methodologies: Capability Maturity Model and More

Some development organizations follow the Capability Maturity Model (CMM), some prefer ISO 9000 recommendations, and still others use Extreme Programming (XP) or Six Sigma methodology. None of these models suggests that it is an exhaustive recipe that covers all cases. The development process that fits your team and project is going to be built and enhanced by you. The process rules, or steps, should allow flexibility; they cannot be "final," in Java terminology.

An Extremely Brief Overview of CMM

The CMM [1] can be used as the basis for diagnosing an organization's software processes, establishing priorities, and acting on them. One of the most important goals of CMM (and of ISO 9000) is the capacity to measure success and reliability of software processes. Measurement results allow for process improvements. Process improvements lead to a maturity level increase. CMM recognizes five levels of maturity in a software development process. Each level comprises a set of process goals that, when satisfied, stabilize an important component of the software process.

1. *Initial*. The process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics. Delivery dates for projects of similar size are unpredictable and vary widely.
2. *Repeatable*. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. *Defined*. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
4. *Managed*. Detailed measures of the software process and product quality are collected. Both the software process and the products are quantitatively understood and controlled.
5. *Optimizing*. Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

The International Organization for Standardization provides an ISO 9000 document that serves as a standard to certify processes and procedures. Whereas CMM focuses strictly on software development, ISO 9000 has a much broader scope: hardware, software, processed materials, and services. CMM provides many more details and measurement criteria.

The bottom line of both methods includes "document before acting," "plan and follow," or "say what you do, and do what you say."

Extreme Programming: Rules of the Game

Extreme Programming methodology gains more popularity every year. This section provides an overview of key XP concepts. (I can almost hear an impatient programmer's voice saying, "Another extremely brief overview? When are we going to see source code?")

Software development is not just about coding, as the art of painting is not just about the right paint selection. The heart of XP is a team of developers able and willing to communicate, plan, and design before coding. The team owns projects and processes and has fun planning and playing the project game by its own rules ("establish and follow"). A subset of XP rules is presented in Figs. 1.1 and 1.2. Does this look like XP?

Six Sigma

Six Sigma is another method that focuses on measurements of process capabilities and offers an integrated approach to shareholder value creation. Six Sigma suggests establishing seven to twelve measures in two areas: "critical to your business" (efficiency measures) and "critical to your client" (effectiveness measures). We select important measurement points that significantly contribute to overall business success.

Now we can consider an equation where business success depends on the measured functions. Some of them can serve as leading indicators that invite discussions such as "Why are these measures not moving in the right direction?" This enhances both discussions and critical thinking. Six Sigma helps to effectively break down complex processes into a matrix with multiple components and consistently works on component improvements.

Rational Unified Process

Rational Unified Process (RUP) is a method and a tool developed by Rational Company (currently a part of IBM, Inc.) that describes development as a four-phase process. In the *inception phase*, developers define the business outcome of the project. The *elaboration phase* considers basic technology and architecture. Developers deal with detailed design and source code in the *construction phase* and finally deploy the system in *the transition phase*. RUP offers software products that support developers as they walk through the phases.

Who implements the rules of the development process? People. This is the bottom line: You can select any methodology that fits your organization. Teams with their managers, leaders, and developers who understand and share the ideas of the methodology will make it work.

Distributed Collaborative Development

It is not an easy task to establish and follow a system, even for a single company. Can it work for a collaboration among several teams? Will this change the rules and the process?

Software communities working on common projects have recently suggested several answers to these questions. Two of the best examples are open source communities and the Java Community Process. However, there are still several factors that limit the success of collaborative engineering today.

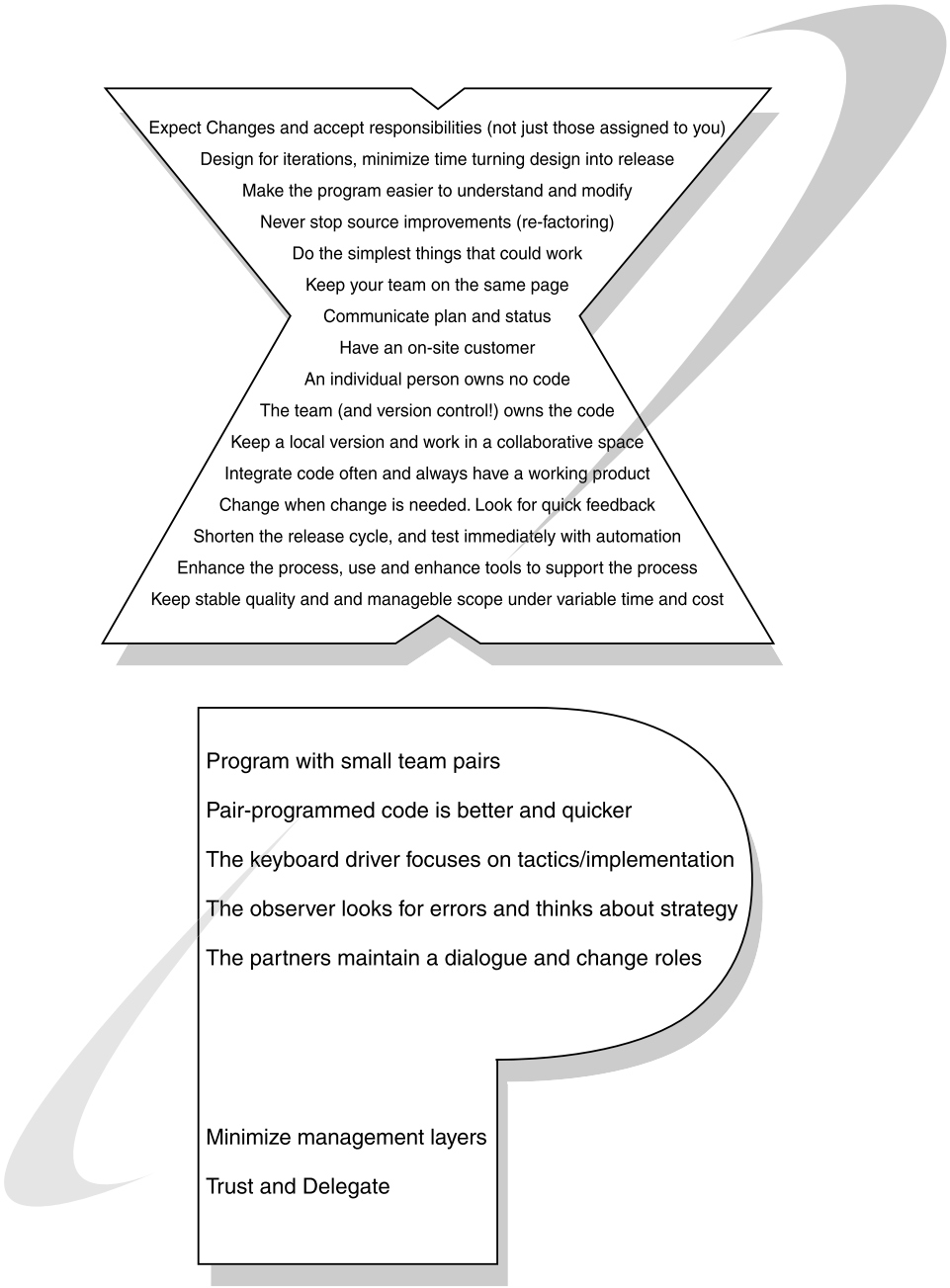

FIGURE 1.1. The Rules of the Game (XP).

The motivation for sharing is very low, and there is no flexibility in establishing the rules of the game: security, roles, access, and so on. The value of the data to be shared or exchanged is unknown, and a contributor rarely gets her or his reward because the contribution itself is not accountable.

For more than a decade, I have worked with remote students and development groups in trying to overcome these limitations. I came up with several innovations, described in a corresponding patent application [2] as distributed active knowledge and process technologies (DKT). DKT supports the process of distributed development and helps to improve processes while enhancing technologies.

Distributed Knowledge Technologies

Existing on-line collaborative services allow users to share only a limited set of data types, usually restricted to messages and files, with the rare addition of a shared organizer or other similar service. This narrows collaborative actions to a small number of fields and introduces limitations on the scope of possible collaboration and data sharing. Though some users are satisfied with restricting their collaborative efforts to sharing files and sending group messages, such systems are often insufficient in scope to allow for efficient workflow in a real collaborative setting.

Existing services on the Internet also limit their collaborative structure to data objects and exclude processes. As a result, the large amounts of data that can accumulate in a group knowledge base cannot be mapped to better processing methods. As the number of data objects increases, it becomes more and more difficult to use the information contained in them to efficiently accomplish goals.

Current system structures do not permit users to collaboratively add objects of unknown data types and a service to process a particular type of data to best suit the goals of a group. They also prevent the creation and implementation of preprogrammed processes, services, or scenarios for distributed processing. This further limits collaborative efficiency.

Existing systems own and fully control their collaborative environments. This limits collaboration to a single system and does not permit systems to share data or other system elements. Data, process, and service sharing between systems belonging to different organizations is an even more complicated issue because there currently is no way for a system to determine and specify elements appropriate for free public sharing, elements that are to be shared on a pay-per-use basis, and elements that are to be exchanged for others of equal value.

Last, current online collaboration is limited by the unwillingness of users to share their data. Even in a collaborative setting, users rarely desire to make their data available to all members of their group, and they make adequate security a condition for sharing information.

The backbone of any online collaborative effort is security, and the current methods of assigning access privileges as a way to make specified data objects available to the appropriate viewers are inadequate.

Existing systems allow limited role-based privileges for all collaborative data. A common system has a limited number of privilege levels (in most cases, two). In such a system, if a user's profile defines her as an "administrator," she has read, write, and delete access to all group data.

If a user is defined as a "member," he can read and add messages but cannot edit or delete existing messages. This kind of system is limiting and does not encourage data sharing because it does not give users control over their data. Users cannot create new custom roles on the fly, cannot select who has certain kinds of access to the information they choose to share, and must provide the same level of access to all members within a privilege class.

The willingness of users to share is also limited by their knowledge of other elements inside and outside the user's system and of their values. It is important to know how valuable a

particular service or data object is from a user's point of view. Current systems provide only the number of times a file has been downloaded and selected positive comments by current users. A new mechanism is needed to provide and update more meaningful usage and viewer response information inside the system and between systems.

DKT covers a need that exists for collaborative systems and permits increased flexibility in the types of data and services that can be shared. It allows data, processes, and services to be created and modified within the same collaborative framework and permits the mapping of appropriate data to these processes. DKT systems can notify interested parties on available objects, processes, and services and provide dynamic evaluation of data and services based on their usage.

DKT systems can negotiate several forms of collaboration and establish rules of access for internal members and outsiders (e.g., pay-per-use, and value-based exchange), using sufficiently flexible levels of data security to foster online collaboration. Sharing can be an enjoyable habit and an eye- and mind-opening experience (not a medical term).

This new mechanism to value data and services creates an environment of active participation in which each contribution is accountable. This accountability motivates contributors to enter this new marketplace, where knowledge and services play the role of virtual currency. There are many ways of transforming virtual currency to hard currency. What is the exact mechanism of this transition? Participants will find or invent a good one.

There are plenty of "how to" details and usage descriptions of DTK systems that might be too boring in the context of this book. They are very well explained in the DKT patent application published by the U.S. Patent and Trademark Office [2].

24×7 Distributed Development Practices

Especially important for an international development team with members distributed over the globe and working twenty-four hours a day, seven days a week, are "24×7" distributed development practices.

Is it possible to achieve coordination, understanding, and cooperation among multiple teams working in different time zones, speaking different languages, and living in different cultures? Can distributed development be cost effective and highly efficient at the same time?

Our virtual team was created as a side effect of several years of on-line training. It was a very attractive concept: to extend time and resources on a global scale. Learning the ABCs of important team member qualities/abilities and developing a core of "black belts" (in Six Sigma terms) for each team was, though very beneficial, quite a challenging task.

Collaborative technologies make the development process highly visible for every team member as well as for the management team. This visibility helps establish and reinforce teamwork rules. The system forces developers to provide daily and weekly plans and status reports that improve analysis and design quality. I say "forces" because the plan and status jobs are the least loved by developers. DKT makes them unavoidable habits. A notification engine working with the task management systems provides an action-reminder if a needed action has not yet been taken.

Global collaboration encourages open expertise exchange under a partnership umbrella. A very valuable benefit is that we can achieve much more working together than we can achieve separately. Network specialists from Hong Kong; a "Fifth Dimension" scientific group from Scandinavia; SZMA (oil and gas factory automation international integrators) engineers;

computer science researchers from St. Petersburg University and Institute of Information Technology in Russia; JavaSchool students and consultants.: these are some examples of teams collaborated in the distributed knowledge environment.

This book includes design and code samples and offers a tool that allows you to build your own system powered by a knowledge engine, support your own development process, establish your own rules of sharing, and, if you wish, connect to other systems and join in collaborative engineering.

Steps in the Process

What Is Your Development Mode: Proactive or Reactive?

We are coming back from the heights of management to the specific steps of the development process. Here is a quick quiz related to your organization's habits toward time distribution on a software project.

Which line (1, 2, or 3) represents the percentage of project time your organization allocates to analysis, design, code, and testing?

	Analysis	Design	Code	Testing
1.	40	30	20	10
2.	10	20	30	40
3.	25	25	25	25

There is no right or wrong answer to this quiz. Your selection sheds some light on your organization's development habits. My experience shows that for bigger projects, shifting gears to line 1 helps to improve quality. However, for really small projects, number 3 works quite well. I have noticed that the second answer is often, though not always, associated with start-ups or organizations that are on the learning curve using "let-me-try-it" paths. (I hope I do not sound critical.)

Amazingly enough, all three ways can work very well for a team that uses one of them as part of a repeatable development process. Such a team usually understands where steps can start and stop and is able to recognize cases in which it makes sense to move along in the cycle. There is no single good answer. This is about your organization development style.

The object-oriented approach to application development is currently the leading approach in the software industry.

I will describe some basic steps of software application development with this approach; I have to add that the steps and their sequence can vary. I am sharing my experience, and I know that there is more than one good methodology, as there is more than one good solution to almost any problem. The most important thing is to have a system and consistently follow it: "Plan and follow" or "say what you do and do what you say."

Basic Development Roles

Developers (or a single developer for a small organization or a small project) play different development roles at major phases of the development process. System definition is usually performed by **system analysts**, **subject matter experts**, and **architects**. Architects and

designers are responsible for system analysis and design. Developers participate in the implementation and deployment phases. **System developers** or service providers are different from **application developers**, who use provider services to create a higher-level system for end users. This separation is not always obvious. Service providers also have end users, but in most cases, their end users are application developers who can use their services.

It is possible that the deployment process requires **system integrators**. This role might disappear as more integration-ready products come out after this book is published. (I know you are smiling.)

Architects do not participate in the deployment process but are responsible for the deployment plan, which is usually expressed in the deployment diagram.

This is a very generic definition that can be customized to fit your development process specifics. For example, according to Sun Microsystems recommendations, Java 2 Enterprise Edition (J2EE)-based development involves the following roles:

- **Tool provider** (for example, Borland): provides development tools
- **Application component provider**: a generic name for Web component providers that develop Java server pages (JSPs), tag libraries, servlets, and related business classes located in the Web container; Enterprise Java Bean (EJB) developers; and Web page designers
- **Application assembler**: uses components developed by application component providers to assemble them into a J2EE application component
- **Deployer**: responsible for application deployment in a specific environment
- **J2EE product provider**: implements a J2EE product
- **System administrator**: provides the configuration and management of the system
- **Quality assurance (QA) engineer**: provides integrated testing.

Most of the development methods obligate developers to provide "unit testing" of the part delivered by the developer to a current system release. But system-level testing often requires a special environment that emulates production and special efforts.

The Role of the Software Architect

Software architecture is the highest level of a software system definition. A **software architect** is the one who actually provides this definition. The architect starts with user requirements and separates them into two categories: **functional requirements** that refer to system tasks and **nonfunctional requirements** that refer to system efficiency, scalability, flexibility, and other "-ilities." The architect participates (to some degree) in the analysis and design processes but does not go into implementation details; he or she focuses on the larger picture of the system instead.

It is important for the architect to negotiate with the client about the metrics (not just words) and priorities of requirements criteria. For example, the "must be scalable" requirement should come with some rough range numbers (e.g., expected number of users).

Basic Steps of the Development Process with an Object-Oriented Approach

The basic steps of the development process are shown in Fig. 1.3. **The rest of the chapter carefully moves over these steps**.